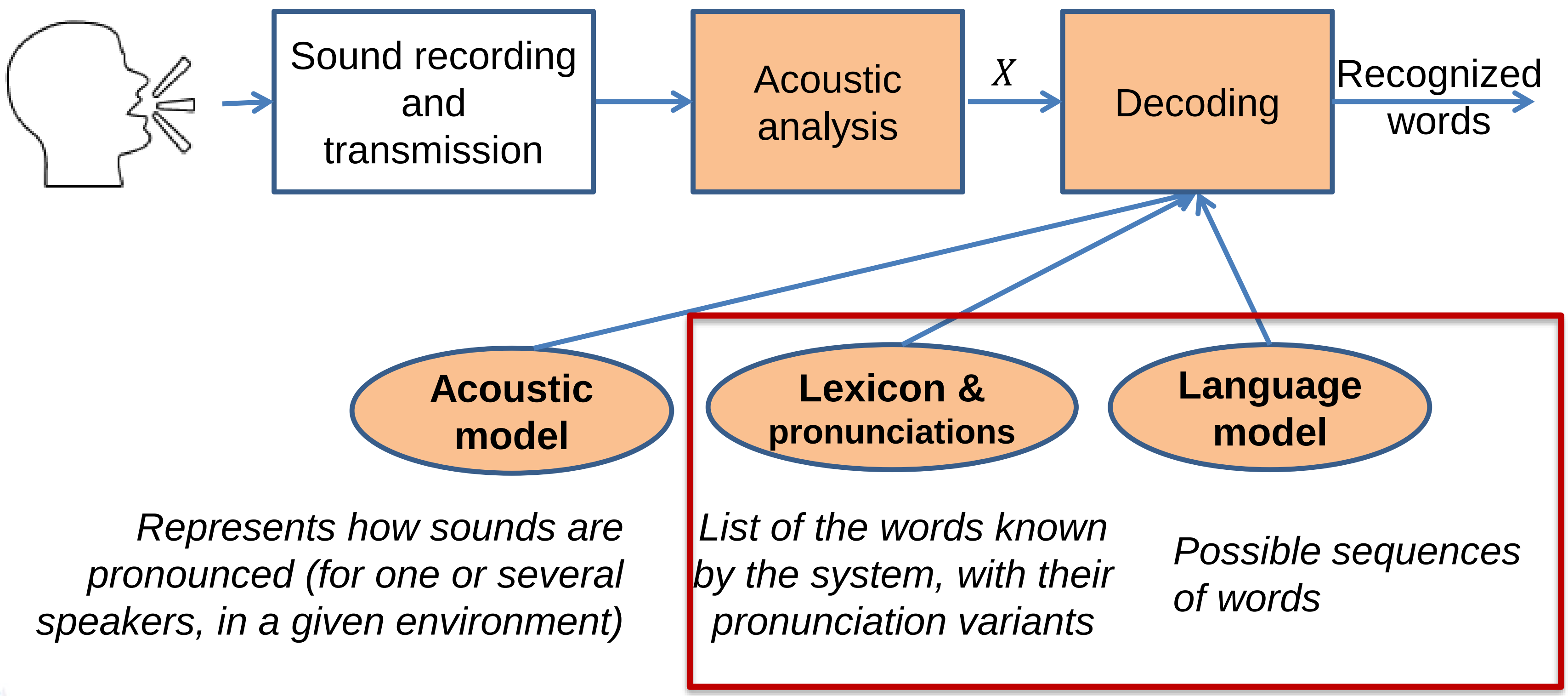
Speech modeling and automatic speech recognition

Lexicons and language models for ASR using pocketsphinx

- experiments –

Romain Serizel (originally from Denis Jouvet)
LORIA – INRIA - Nancy

Automatic speech recognition



Lexicon

- Specify a list of words and their pronunciation variants

- Example (excerpt from pocketsphinx lexicon)

```
one            HH W AH N
one(2)         W AH N
person         P ER S AH N
quarter        K AO R T ER
quarter(2)     K W AO R T ER
quarters       K W AO R T ER Z
quit           K W IH T
```

word

pronunciation

numbering
of pronunciation
variants

Language models

- Provide information on the possible sequences of words

- Context-free grammars
  - Specify sequences of words corresponding to « sentences » (i.e. global constraints)

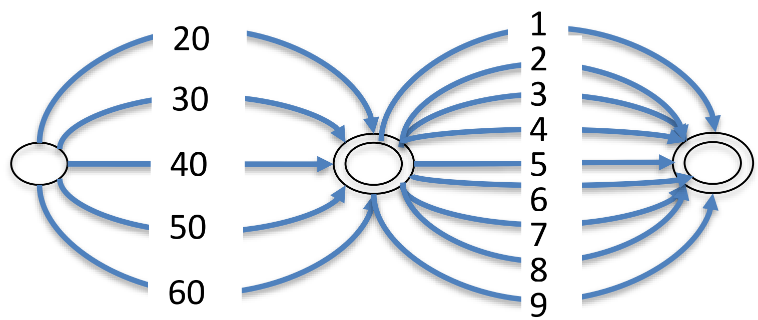
- n-gram statistical model
  - Provide local constraints (on sequences of *n* words)
  - Estimation from text corpora

- Neural network models
  - Many different approaches proposed in the literature

Context-free grammars

- Rules (or graphs) that describe exactly the allowed sentences (sequences of words) of the language (e.g., isolated digits, numbers, …)
  - Ex. Number_from_20_to_69

    Number_from_20_to_69 = Tens_from_20_to_60
    | Tens_from_20_to_60 . Units_from_1_to_9 ;
    Tens_from_20_to_60 = 20 | 30 | 40 | 50 | 60 ;
    Units_from_1_to_9 = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ;



- Global constraints on the sentences
- Complex and difficult to define for large vocabularies
- Do not allow the recognition of sentences that do not respect the grammar

jsgf ⇔ JSpeech Grammar Format

- *The JSpeech Grammar Format (JSGF) is a platform-independent, vendor-independent textual representation of grammars for use in speech recognition*

- Example of JSGF grammar

```
#JSGF V1.0;
/**
 * JSGF Grammar for Turtle example
 */
grammar goforward;
public <move> = go forward ten meters;
public <move2> = go <direction> <distance> [meter | meters];
<direction> = forward | backward;
<distance> = one | two | three | four | five | six | seven |
eight | nine | ten;
```

N-gram language model

- Statistical model

- Probability of a word sequence

$$P(w_1, \ldots, w_N) = P(w_1) \prod_{i=2..N} P(w_i | w_1, \ldots, w_{i-1})$$

- N-gram approximation, as n-grams deal with sequences of n words

$$P(w_i | w_1, \ldots, w_{i-1}) \triangleq P\left(w_i \middle| w_{i-(n-1)}, \ldots, w_{i-1}\right)$$

- N-gram parameters are computed from large text corpora

Examples of n-grams

- Sentence : *the sky is blue*

  ➔ *<s> the sky is blue </s>* including start and end of sentence symbols

- Unigrams

  P(Sentence) = P(*the*) . P(*sky*) . P(*is*) . P(*blue*)

- Bigrams

  P(Sentence) = P(*the*|*<s>*) . P(*sky*|*the*) . P(*is*|*sky*) . P(*blue*|*is*) . P(*</s>*|*blue*)

$$\mathrm{P(Sentence)} = \mathrm{P}(w_1| < s >) \prod_{i=2,..N} \mathrm{P}(w_i|w_{i-1})$$

- Trigrams

  P(Sentence) = P(*the*|*<s>*) . P(*sky*|*<s> the*) . P(*is*|*the sky*) . P(*blue*|*sky is*) . P(*</s>*|*is blue*)

Pocketsphinx

- *PocketSphinx is a lightweight speech recognition engine, specifically tuned for handheld and mobile devices, though it works equally well on the desktop*
- https://github.com/cmusphinx/pocketsphinx
- Available as a python package – thus easy to install and to use
- Remark:
  - The speech recognition performance depends on the quality of acoustic models (and on the adequation of the model with the test data)

Lexicons and language models experiments

- Lexicon and language model experiments
  - Pocketsphinx generic lexicon and language model (i.e., ngram)
  - Digit loop grammar (to write)
  - Grammar for sequences of digits of known length (to write)
- Speech corpus for performance evaluation
  - Sequences of digits (1 digit, 3 digits & 5 digits)
  - With added noise: signal to noise ratio (SNR) of 35 dB, 25 dB, 15 dB and 05 dB
- Performance evaluation to do – estimate word error rate with respect to
  - Language model
  - Length of digit sequence
  - Speaker group
  - Signal to noise ratio

Digit corpus

In file « `td_corpus_digits.zip` »

```
td_corpus_digits
    SNR35dB
        man
            seq1digit_200_files
            seq3digits_100_files
            seq5digits_100_files
        woman
        boy
        girl
    SNR25dB
    SNR15dB
    SNR05dB
```

td_corpus_digits
- SNR35dB
  - man
    - `seq1digit_200_files` ➔ files *.wav & *.ref of isolated digits
    - `seq3digits_100_files` ➔ files *.wav & *.ref of 3 digit sequences
    - `seq5digits_100_files` ➔ files *.wav & *.ref of 5 digit sequences
  - woman
    - *Same organization as for « man »*
  - boy
    - *Same organization as for « man »*
  - girl
    - *Same organization as for « man »*
- SNR25dB
  - *Only « man » data; isolated digits, and 3 and 5 digit sequences*
- SNR15dB
  - *Only « man » data; isolated digits, and 3 and 5 digit sequences*
- SNR05dB
  - *Only « man » data; isolated digits, and 3 and 5 digit sequences*

Lexicons & language models

- In file « `ps_data.zip` »

```
ps_data
    model
        en_us                  → English acoustic model
    lex
        cmudict-en-us.dict     → English generic lexicon
        turtle.dic             → Small lexicon (for the jsgf grammar)
    lm
        en-us.lm.bin           → Generic english Ngram model
    jsgf
        goforward.gram         → Example of jsgf grammar
    exemple
        goforward.raw          → Speech file
```

Examples of usage of pocketsphinx

In file « `ps_exemples.zip` »

`ps_exemples`

- `decoder_ngram.py` (https://raw.githubusercontent.com/cmusphinx/pocketsphinx/master/swig/python/test/decoder_test.py)
  - From web – generic English lexicon and language model
  - Slightly modified to use lexicon and models from directory ps_data
  - Send speech data to decoder in small blocks
- `decoder_jsgf.py` (https://raw.githubusercontent.com/cmusphinx/pocketsphinx/master/swig/python/test/jsgf_test.py)
  - From web – use a jsgf grammar
  - Slightly modified to use lexicon and models from directory ps_data
  - Send speech data to decoder in small blocks
- `decoder_utt_ngram.py`
  - Modified version of `decoder_ngram.py` that sends the whole speech file in a single call
- `decoder_utt_jsgf.py`
  - Modified version of `decoder_jsgf.py py` that sends the whole speech file in a single call

Setup

Here, assume running in a python virtual environment

```
python3 -m venv asr-env
source asr-env/bin/activate
```

→ Create python virtual environment
→ Activate virtual environment

```
pip install pocketsphinx
    python ps_exemples/decoder_ngram.py
    python ps_exemples/decoder_utt_ngram.py
    python ps_exemples/decoder_jsgf.py
    python ps_exemples/decoder_utt_jsgf.py
```

→ Install pocketsphinx
→ Check that program runs
→ Check that program runs
→ Check that program runs
→ Check that program runs

```
pip install asr-evaluation
```

(usage: `wer -i toto.ref toto.hyp`) → To compute word error rate

….. → Run the experiments…

`Deactivate` → Exit the virtual environment

Example of code using pocketsphinx `decoder_ngram.py`

```
# Create a decoder with certain model
config = Decoder.default_config()
config.set_string('-hmm',  'ps_data/model/en-us')
config.set_string('-lm',   'ps_data/lm/en-us.lm.bin')
config.set_string('-dict', 'ps_data/lex/cmudict-en-us.dict')

# Decode streaming data.
decoder = Decoder(config)

decoder.start_utt()
stream = open('ps_data/exemple/goforward.raw', 'rb')
while True:
     buf = stream.read(1024)
     if buf:
          decoder.process_raw(buf, False, False)
     else:
          break
decoder.end_utt()

hypothesis = decoder.hyp()
print ('Best hypothesis: ', hypothesis.hypstr)
```

Computation of word error rate

- Need two files
  - `Data.ref` ⬄ for the reference (i.e., transcriptions of the speech data)
  - `Data.hyp` ⬄ for the ASR hypothèses (i.e., speech recognition output)
- Warning – **files must be aligned**:
  - The n-th line of `Data.ref` must correspond to the n-th line of `Data.hyp`
    i.e., reference for n-th speech file, and associated speech recognition output

- Example
  - `Data.ref`
    ```
    eight four five
    five four seven
    zero seven six
    oh one three
    four five one
    …
    ```
  - `Data.hyp`
    ```
    eight eight four five
    five four seven
    zero seven six
    five oh one three
    four five one
    …
    ```

Computation of word error rate (2)

Use the python module `wer.py`

```
wer -i Data.ref Data.hyp
```

Example of result

```
REF: ***** eight four five
HYP: EIGHT eight four five
SENTENCE 1
Correct          = 100.0%    3   (      3)
Errors           =  33.3%    1   (      3)
REF: five four seven
HYP: five four seven
SENTENCE 2
Correct          = 100.0%    3   (      3)
Errors           =   0.0%    0   (      3)
REF: zero seven six
HYP: zero seven six
SENTENCE 3
Correct          = 100.0%    3   (      3)
Errors           =   0.0%    0   (      3)
REF: **** oh one three
HYP: FIVE oh one three
SENTENCE 4
Correct          = 100.0%    3   (      3)
Errors           =  33.3%    1   (      3)
```

Estimated word error rate and confidence interval

The end of the file produced by the module `wer` looks like

```
Sentence count: 40
WER:      19.167% (        23 /       120)
WRR:      99.167% (       119 /       120)
SER:     100.000% (        40 /        40)
```

Word error rate — Number of errors — Number of word occurrences

Confidence intervals ⇔ uncertainty on the estimated word error rate

95% confidence interval :

$$1.96\sqrt{\frac{P\,.(1-P)}{N}}$$

where P is the word error rate, and N the number of words in the test set

Here:

$1.96\sqrt{\frac{0.19\,.(1-0.19)}{120}} = 0.070$ that is a word error rate of 19.2%±7.0%

Preparing lexicons and language models

- Lexicon corresponding to English digits, including ‘oh’
  - Lexicon: *zero, one, …, nine, oh*
  - Extract words and associated pronunciations from the pocketsphinx lexicon (don’t forget pronunciation variants)

- Digit loop jsgf grammar
  - Grammar allowing sequences of digits of any length

- Jsgf grammar for digit sequences of known length
  - 3 entry points corresponding to sequences of 1 digit, 3 digits and 5 digits
  - Can be specified in the same jsgf grammar file

Adaptation of program examples for the experiments

1. Adapt the programs to handle the different jsgf format digit grammars

2. 3 programs to write and test on one or several speech files (digit sequences)
   - One using generic ngram language model
   - One using the digit loop grammar
   - One using a digit sequence of known length (of course, matching the length of the digit sequence to recognize)

3. The make another modification to handle a set of speech files, and to write the speech recognition results in an output text file, for example

```
temp/digits/raw/SNR05dB/boy/seq3digits_40_files/SNR05dB_boy_seq3digits_040.raw :: five
temp/digits/raw/SNR05dB/boy/seq3digits_40_files/SNR05dB_boy_seq3digits_022.raw :: oh five eight eight
temp/digits/raw/SNR05dB/boy/seq3digits_40_files/SNR05dB_boy_seq3digits_023.raw :: zero eight oh three oh
temp/digits/raw/SNR05dB/boy/seq3digits_40_files/SNR05dB_boy_seq3digits_024.raw :: one nine three five
temp/digits/raw/SNR05dB/boy/seq3digits_40_files/SNR05dB_boy_seq3digits_025.raw :: nine eight six
```

Estimating word error rates

1. Make a script (program) that reads an output text file, and creates
   1. A reference file « `Data.ref` » by getting the reference transcript of each speech file
   2. An ASR hypothesis file « `Data.hyp` » that contain the speech recognition results

2. Apply the program computing the speech recognition errors

   ```
   wer -i  Data.ref  Data.hyp  >  Data.results
   ```

3. Look at the content of the output file « `Data.results` »

4. Extract the word error rate on the data set,
   and **compute the corresponding 95% confidence interval**

1 – Impact of the language model

- Data
  - SNR35dB ⇔ very low noise level
  - man ⇔ man speakers
  - 1 digit, 3 digits & 5 digits ⇔ i.e., 400 files

- Language models
  - Generic ngram language model
  - Digit loop grammar in jsgf format
  - Digit sequence of known length (1 digit or 3 digits or 5 digits) in jsgf format

- Evaluation to do
  - **For each language model**, compute the word error rate and the associated 95% confidence interval on the 400 speech files corresponding to man speakers and very low noise level (35 dB SNR)

2 – Variability with respect to speaker groups

- Data
  - SNR35dB ⇔ very low noise level
  - man, woman, boy, girl ⇔ i.e., all the four speaker groups
  - 1 digit, 3 digits & 5 digits ⇔ i.e., 400 speech files per speaker group

- Language model
  - Jsgf grammar corresponding to sequences of digits of know length (1 digit or 3 digits or 5 digits)

- Evaluation to do
  - **For each speaker group** *(man, woman, boy, girl),* compute the word error rate and the associated 95% confidence interval on the 400 speech files corresponding to each speaker group, and very low noise level (35 dB SNR)

3 – Performance with respect to the length of the digit sequence

- Data
  - SNR35dB ⇔ very low noise level
  - man, woman ⇔ i.e., only adult speech data
  - 1 digit, 3 digits & 5 digits ⇔ i.e., 400 files per speaker group

- Language model
  - Jsgf grammar corresponding to sequences of digits of know length (1 digit or 3 digits or 5 digits)

- Evaluation to do
  - **For each length of digit sequences** *(1 digit, 3 digits, 5 digits*), compute the word error rate and the associated 95% confidence interval on the speech files corresponding to each category (400 files for 1 digit, 200 files for 3 & 5 digit sequences), and very low noise level (35 dB SNR)

4 – Impact of the noise level

- Data
  - SNR35dB, SNR25dB, SNR15dB & SNR05dB
  - man ⇔ i.e., only man speakers
  - 1 digit, 3 digits & 5 digits ⇔ i.e., 400 speech files

- Language model
  - Jsgf grammar corresponding to sequences of digits of know length (1 digit or 3 digits or 5 digits)

- Evaluation to do
  - **For each signal to noise ratio** (SNR35dB, SNR25dB, SNR15dB & SNR05dB), compute the word error rate and the associated 95% confidence interval on the 400 speech files corresponding to each SNR category

Data and results to return

(Preferably by uploading zip files on ARCHE (UL ENT web site), or else by mail, if any problem)

- Per group (if you work by groups of at most two people), or individually
  - The programs you have written and used, as well as grammar files and lexicons
  - The program output corresponding to the various speech recognition experiments, and to the computation of the word error rates (output of program `wer`)
  - A short document (2 to 3 pages) summarizing the experiments, and presenting and discussing the results (WERs of the four previous experiments)